General control-theoretical framework for online resource allocation in computing systems

Alessandro Vittorio Papadopoulos[a], Martina Maggio[b],
Sara Negro[a] and Alberto Leva[a,*]
[a]Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
[b]Department of Automatic Control, Lund University, Sweden
* Corresponding author, leva@elet.polimi.it

Abstract

System-theoretical methods are already used for the control of computing systems, but much more can be done exploiting said methods for their *design*. This requires to express in control-theoretical terms desires and specifications that originate in the computer science domain, which may not be immediate. It also requires to accept that part of the addressed system be modified, which may pose some acceptance problems. However, if those issues are handled correctly, the payback is often very relevant. This paper demonstrates the above ideas in the context of resource allocation.

Keywords: Computing Systems Design; Feedback Control.

1 Introduction

In recent years, a significant convergence is observed between the need for "self adaptation", evidenced in the computer science community, and

the possible responses coming from methodologies developed in the control one [14]. Examples are found in various areas, from security [11, 12] to Quality of Service [1, 32, 34, 35], and more. Both communities agree on the potential of that convergence [14], which seems very *natural*, as some important computing system components – think for example of a scheduler – are controllers in nature [26].

Curiously enough, however, virtually all the available literature on feedback control of computing systems only addresses the particular problem of starting from a fully functional system and adding some external loops to adapt its parameters, think for example about the queue length of a web server [13]. Doing so means closing a loop around a system that already contains some control. This control, generally designed directly as algorithms and not by means of dynamic systems, can be shown to be basically a useless complication when designing the outer loop. In this work we propose to shift the perspective from the control of a fully functional system to the design of part of that system as a controller.

This perspective shift involves isolating some "core physical phenomenon", removing parts of the system that attempt to control it in a non system-theoretical (i.e., essentially heuristic and hard to model) manner, and replace them with properly designed controllers. As the examples later on will clarify, doing so often allows simple paradigms to describe the control problem in its entirety.

This paper proposes a general methodology to introduce computing systems self-adaptation in resource allocation problems by direct application of simple feedback control techniques. As will emerge in the following, quite a numerous variety of problems appear to be addressable with the proposed ideas, but of course there are some applicability limits. For example, some

computing systems-related problems cannot be handled without the introduction of event-based models, and if this is the case, the proposed approach is not fit to the problem. However, to stick to the same example, in more than one case the event-based paradigm is brought in without a real necessity for it. If this is conversely the case, the proposed approach is advantageous.

The methodology was implemented in the Linux operating system, and experimental results are presented. The methodology is named SMART – which stands for Sensing, Modelling, Actuating, Regulating and Tuning.

2 Related work

This work deals with a feedback control based resource allocation mechanism. The allocator resides completely at the software level, differentiating the contribution from hardware ones like [2, 6, 10]. Control techniques have been used also at the software level to provide performance [3, 21, 30, 31, 36] and reliability [8] in web servers. Real-time schedulers have been complemented with adaptation ability [7, 13, 22] and operating systems were also the target for self-aware implementations [9, 18, 19, 29]. The difference between these work and our proposal lies in generality, since we are not solving a specific problem as usually done, e.g., web server control. However, we recognise that generality could come at the cost of a smaller improvement in application that could be better controlled with a specific control policy.

At the same time, system level resources were the target of many optimisations. For example, machine learning was used for managing a memory controller [17] and a neural network is implemented as a dedicated hardware to manage resource allocation in multicore chips [6]. While these approaches allow system level adaptation to be performed without input from the application programmer, application performance must be inferred from either

low-level metrics like performance counters [2] or high-level metrics like total system throughput [6] and there is no way for the system to understand if a specific application is meeting its goal.

3 A preliminary example

To help perceive the *rationale* of SMART, a brief example based on [25, 27] is reported. Consider a multitasking system with fixed-priority preemptive scheduling. The scheduler selects the ready task with the highest priority p_i, assigns the CPU to it for a fixed quantum q, then preempts it, and selects the next task to run. Priorities p_i can be changed by suitable system calls, and measurements of the CPU time consumed by each task are available.

Suppose that the goal is that each task receives a given CPU time percentage. One can leave the existing scheduler in place, and design a feedback controller acting on the priorities, based on the measured CPU use. In this case the loop comprises not only the tasks' behaviour in response to the CPU allocation, but also the operation of the system calls, the queue management logic, and so forth.

This is an evident and useless complications. Control desires are in fact posed on the "core phenomenon" consisting of the tasks' behaviour, which is much simpler than what is enclosed in the loop if the system is taken *as is*. A task receives a certain allotted amount b_i of CPU time or "burst", and returns the CPU after a time that can differ from b_i, typically because the CPU was yielded in advance, or a critical section delayed the preemption.

Denoting by $\tau_i(k)$ the CPU time accumulated by the i-th task at the k-th scheduler intervention, the core phenomenon to control is simply ruled by

$$\tau_i(k) = \tau_i(k-1) + b_i(k-1) + \delta(k-1) \tag{1}$$

where the disturbance δ accounts for the mentioned discrepancies.

Notice also that the way the original scheduler is designed is a further source of complication. In fact, the influence of "who can interrupt who" – i.e., the priorities – on the CPU distribution is not easy to describe and control. If one instead acts on the b_i directly, any goal expressed as a desired behaviour of tasks' CPU use over time, can be achieved in a methodologically grounded manner, and formally assessed. For example, [20] proposes a PI-based mutivariable control scheme, of which [28] describes a microcontroller kernel implementation.

If the proposed perspective shift is not accepted, the authors believe that complexity may become a real issue. Consider for example the Linux scheduler. In Kernel 2.4.37.10, released in September 2010, its code was contained in one file 1397 lines long. In Kernel 2.6.39.4, released in August 2011, the scheduler is spread among 13 files totalling 17598 lines. The scheduler of [28] fits in about 500 lines, only a fraction of which devoted to the control algorithm.

One may object that the Linux scheduler provides features not addressed in by these 500 lines. However, most of the Linux scheduler is devoted to realise alternative policies, that can all be proven to be special cases of a single dynamic system [25], and could be reconciled as different parametrisation of a single controller, or the generation of different set point signals. Also, much of the observed Linux code increase is due to the inclusion of multi-core CPUs, which in fact just proves that controllers conceived directly as algorithms do not scale up as naturally as controllers conceived as dynamic systems: the model in [25], incidentally, already considers multiple cores.

In any case, whenever the code of a core functionality (as the scheduler is, in an operating system) explodes by a factor of ten in one year after

twenty of lifetime, it may be a good idea to reconsider the design approach—for example, in a more system-theoretical way than before. The proposal of SMART comes from noticing that the simplifications just shown in the scheduling case carry over to many other computing system problems, thus suggesting a general design methodology.

4 The SMART Design Framework

The example of Section 3 should evidence the usefulness of structuring computing systems adaptation problems as a control one right from the beginning: if this calls for modifying part of the system, most likely – and a bit crudely – one is just removing "unduly introduced physics". As will be shown, this is in general a benefit.

SMART can be seen as a sequence of design steps. That sequence most likely appears to a control scientist as plain trivial, but nonetheless has at least two merits. First, in the computing system domain, such a design structuring is novel indeed. Second, and most relevant here, SMART shows that some problems for which complex control techniques were (successfully) applied, can be tackled with simpler ones as well if conveniently structured. Since in computers "one creates physics", their management is an impressively suited domain for process/control co-design. In the opinion of the authors, SMART provides a nice contribution in that direction.

Apparently, an accurate analysis on the (expected) impact of the approach on the addressed system must be performed, as in fact the proposed methodology lies and operates in an abstract level. In principle, thus, there some problems may emerge so that a solution can be found at the abstract level, but its implementation is not possible or inconvenient owing to application-specific limitations. Since this work does not just present

the abstract level but also an implementation of it, the mentioned analysis allows to better evidence the effectiveness of such an approach.

For the sake of clarity we will go through the procedure twice. Here, we present the main steps of SMART at a glance, to provide the reader with an overview of the approach. Then, we apply SMART to a representative example (a resource allocation problem), to evidence its advantages, and delve into further considerations that would not be so clear if presented totally *in abstracto*.

The first step of SMART is to formulate the problem as a set point tracking one, a functional minimisation one, or any combination thereof. This indicates the measurements to be acquired from the system to qualify its behaviour (i.e., the **S**ensing part). Also, this dictates which sensors need introducing – if not already present – and where. In fact, SMART is not applicable only if this step cannot be carried out, but no such case has emerged yet. On the other hand, as the example will show, this is the point where some system redesign is most frequently required.

Having defined sensors, one must see if the system allows for some action to influence the chosen measurements of its behaviour. Generally the answer is affirmative, although here too some additions could be suggested. The mentioned actions provide the **A**ctuation part.

Then, a (generally dynamic) **M**odel needs writing to relate A to S. Unfortunately, and contrary to other control domains, said model cannot be grounded on any set of physical principles. However, thanks to the vicinity of A and S to the anticipated "core phenomenon", in many cases of interest simple and synthetic considerations allow to write the required model with some convenient grey box approach. Also, the same vicinity above normally allows for quite simple models—this too is exemplified in the following.

The **R**egulator structure, quite intuitively, can now emerge from that of the model, and thanks to the elimination of undue physics pursued so far, is tendentiously simple, which eases – and sometimes *de facto* enables – the use of powerful control techniques. As a result, the subsequent **T**uning phase quite often leads to computationally light solutions.

The applicability limits of SMART are dictated by the feasibility of the steps above. Fortunately, in many cases the analyst can figure out whether SMART is applicable or not based on a quite preliminary analysis.

5 Resource allocation via SMART

The problem addressed is to complement a multitasking operating system with an application-aware resource allocator, with the purpose of making an application reach its goal not only within a certain time, but also with a prescribed progress rate. The problem is quite general, as it emerges whenever the rate of processed data delivery is of concern. A notable case, for example, is the encoding of video streams.

5.1 Sensing

The system behaviour is quantitatively indicated by the application "progress rate", which however is not currently measured by the operating system *as is* directly. Operating systems typically provide information on what resources an application is engaging, by measuring the used CPU time, the number of assembler instructions executed per second, the cache miss or memory page fault rates, and so forth. This however does not tell whether the application is using resources to do useful work, or for example just spinning on a lock. Many workarounds were attempted to infer the application progress from the taken resources but these are invariantly architecture-specific to

the detriment of portability, and above all miss the core issue: the use of resources ultimately depends on the application, hence that is where sensing must be located.

For sensing, here the recently proposed Application Heartbeats (HB) framework [15] is used. In HB, applications are instrumented to emit a "heartbeat" when something relevant to appreciate their progress is accomplished (for example, a video encoder could signal each frame completion). Using HB is quite simple for the application programmer, who knows what the application is meant for. Also, and most important, doing so naturally locates sensing at the correct level—a small modification for a system, a significant simplification for its control.

5.2 Actuation

The allocator can exert control actions by allotting an application more or less resources. Here too, SMART first requires to observe what the system *as is* provides as possible actuators, limiting however the choice to what acts on the desired behaviour directly. For example, the number of CPU cores and their clock frequencies are suitable actuators, in that their effect mostly (not to say only) depends only on what the application is doing. The CPU time fraction is less suited, as the actual (not the desired) distribution of the CPU also depends on the scheduler operation. Other quantities like the nice number are finally totally inadequate, as their influence is even more indirect than that of the desired CPU time fraction, and difficult to model.

In the case at hand, the number of cores and the frequency are adopted as actuators. This is a reasonable choice as confirmed by the subsequent modelling and control design phases. Hence, for actuating, no intervention on the system is here introduced.

5.3 Modelling

It was stated that, if sensors and actuators are introduced at the right level, i.e., as near as possible to the core phenomenon, the model required for control synthesis "tends to be simple". This is now demonstrated in the addressed problem.

The required model is a MISO dynamic system, having as inputs the actuators' actions $a_i \in \mathbf{a}$, and as output the sensor output $s_j \in \mathbf{s}$. The main problem is a variable and hard to predict actuator efficacy. For example, if an application switches from a CPU-bound to a memory-bound behaviour, the efficacy of a "number of cores" actuator will drop. Such variability occurs at a time scale dictated by the application code, and sometimes the processed data. Only the application itself could notify about actuators' effectiveness, but from a technological viewpoint this is quite unrealistic.

However, in the example two time scales exist: one at the *code level*, where most of the unpredictability resides, the other at the *observed behaviour* level, where sensors' measurements in fact average code level facts over convenient time spans (for example, the number of heartbeats in the last second). The time scale of actuator actions is the code-level one, hence at the observed behaviour level the effect of actuators can be safely regarded as instantaneous, or ruled by very simple dynamics. It is thus reasonable to assume for the model the form

$$\mathbf{s}(k) = \phi\left(\mathbf{s}(k-1), \mathbf{s}(k-2), \ldots; \psi\right) + \gamma\left(\mathbf{a}(k-1), \mathbf{a}(k-2), \ldots; \vartheta\right) \qquad (2)$$

where the discrete time index k counts the measurement (and control) instants, while ψ and ϑ are parameter vectors. A key point, further stemming from the remark above, is that in many cases one can assume $\phi = 0$

(thus no dynamics at the control time scale except for a delay) or at most $\phi = p\,\mathbf{s}(k-1)$, i.e., a first-order dynamics. Moreover, ψ – or p in the simplified case – are more connected to the time span over which sensors measurements average the code-level behaviour than to that behaviour in detail: as a result, p can generally be considered time invariant. Adopting from now on these simplifications, and specialising to the example, the required model can be written as

$$hr(k) = (p)\,hr(k-1) + (1-p)\,\sigma(k-1), \qquad 0 \leq p < 1 \tag{3}$$

where hr is the application heart (beat) rate, and σ represents – in suggestive terms – the "speedup" yielded collectively to the application by the resources allotted through the actuators.

The problem is therefore confined to determining a suitable map $\{a_i\} \mapsto \sigma$, possibly time-varying due to the variability of ϑ. Here too, the problem characteristics help if instrumentation is carried out properly. In fact, actuator efficacy variability is due mostly to the application, and generally undergoes modifications that may be abrupt and of remarkable entity, but sporadic with respect to the control time scale. To explain, first think again to a video encoder. During the management of each frame, it will more or less traverse the same sequence of operations: first read data, being thus bound essentially to some peripheral, then compute, which is basically CPU-bound, and finally write, which is again peripheral-bound. If such an application is instrumented by making it emit a heartbeat per frame, large variability of a "number of cores" actuator efficacy are not to be expected. Consider, conversely, an application that reads a lot of data, and then performs some mathematically-intensive processing of them in a batch fashion.

If the code is written by exploiting an available parallel architecture, the same actuator will be of hardly any efficacy in the read phase, but very relevant in the processing one. If the control time scale is chosen properly, there will be several control steps in both phases, and so a single abrupt system variation will be observed. Clearly in both cases things could complicate a lot if instrumentation is done differently, but this just testifies that instrumenting an application requires knowledge of it.

For completeness, one could object that also external facts (e.g., an unavailable lock) could introduce variability. However, the only major difference with respect to the second example just given is that such facts cannot be forecast even knowing about the application. In any case, with a properly chosen instrumentation and control time scale, the sporadic character of variations still carries over—and anticipating a bit, standard adaptive controllers can handle such situations successfully.

As for the $\{a_i\} \mapsto \sigma$ (static) map, in most cases synthetic considerations are sufficient to devise a structure for it, resorting then to grey box identification for its parametrisation. A quite general form for that map, according to experience, is

$$\sigma = \prod_{i=1}^{N_a} \left(k_i a_i^{\alpha_i} + o_i\right) \tag{4}$$

where N_a is the number of actuators, defining $\vartheta \in \Re^{3N_a}$ as

$$\vartheta = \begin{bmatrix} k_1 & \alpha_1 & o_1 & k_2 & \alpha_2 & \dots & k_{N_a} & \alpha_{N_a} & o_{N_a} \end{bmatrix}' \tag{5}$$

Apparently, if the actuators are the number of cores c and the normalised frequency f, (4) reduces to

$$\sigma = \left(k_c c^{\alpha_c} + o_c\right)\left(k_f f^{\alpha_f} + o_f\right). \tag{6}$$

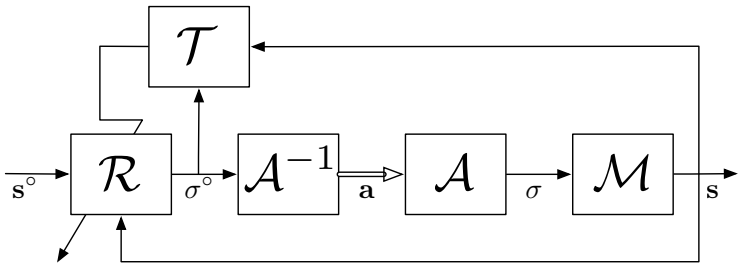


Figure 1: The SMART control scheme.

5.4 Regulating and tuning

The SMART framework naturally leads to a control scheme like that of Figure 1.

The modelling phase has shown that by conveniently instrumenting the system, the problem can be addressed taking as "process" model a MISO Hammerstein one (blocks $\mathcal{A}$ and $\mathcal{M}$). There is a vast literature on such models, but the situation (abrupt but sporadic variabilities of otherwise smooth models with very simple dynamics) and the typical requirements of computing systems control/design (high speed and computational lightness) advise here too for a domain-specific approach. In other words, the choice is made to reason for the control synthesis as if the manipulated variable was a desired speedup σ°, while at the same time taking profit of the degrees of freedom introduced by the $\sigma^\circ \mapsto \{a_i\}$ (i.e., block $\mathcal{A}^{-1}$ in Figure 1) as part of the system components' design.

Quite intuitively, in fact, a given speedup value will not be yielded by a single actuator combination. If that is the case, to select the combination output by the $\sigma^\circ \mapsto \{a_i\}$ for a certain value of σ°, one can bring in for example considerations regarding the consumed power, the least use of some or some other resources based on what are most "precious" for the system, or technologically simpler to act upon, and so forth. Any such mechanism

will work under the assumption that $\sigma^\circ \mapsto \{a_i\}$ is either known or estimated reliably enough, which means that the **R** phase works if the **A** one is carried out properly.

Here too, the addressed example is useful to clarify. The number of cores and the frequency are used as the actuators. The number of cores can assume only integer values, but can be treated as real thanks to a Pulse Width Modulation-like policy. For example, if 2.4 cores must be assigned to an application, the actuation policy allots 3 cores for the 40% of the control interval and 2 cores for the 60% of the control interval. The two actuators however impact the system differently, since in general the frequency is the same for all cores. Thus a frequency change is more invasive than a core re-allocation, and based on that (design) consideration, the adopted solution can be summarised as follows:

1. starting from the speedup σ° computed by the controller, c is computed maintaining the clock frequency of the previous step,

2. if c is not within its saturation values, the frequency is increased or decreased (accordingly to which saturation is violated) and step 1 is repeated,

3. otherwise the computed number of cores and clock frequency are assigned to the application.

Given all the above, the chosen $\sigma^\circ \mapsto \{a_i\}$ map and the cascaded Hammerstein MISO model composed of (3) and (4), can be viewed as a time-varying linear SISO model. The *nominal* $\sigma^\circ \mapsto \{a_i\}$ map above is physically realised in the system as block $\mathcal{A}^{-1}$ in Figure 1 and the task of counteracting variabilities is delegated to an adaptive controller designed in a linear context. In so doing, the considered problem (like many others) can be cast in

the strong framework of linear adaptive Model Predictive Control (MPC), to which it did *not* belong prior to its structuring along the SMART approach.

The presented resource allocation example can be treated with an adaptive model predictive controller, based on an ARX-type process model of fixed structure, parametrised on line by recursive identification. The choice of the ARX orders, the prediction horizon, and the two weights on the error and the control energy, are the subject of the subsequent **T**uning phase, that thanks to the previous problem structuring, is completely standard.

6 Results

This section presents some results for the resource allocation case. Two tests are reported, referring to two applications of the PARSEC benchmark suite [5]. The first example shows that SMART leads to satisfactory results with applications that are not easy to instrument. The second example conversely refers to an easy to instrument application, which is controlled with multiple actuators, and shows that also in this case adaptive controllers designed with the SMART approach are simpler than in its absence. Sections 6.1 and 6.2 report simulation tests while experimental ones are presented in Section 6.3.

6.1 Example 1 - vips

Vips is based on the VASARI Image Processing System and includes fundamental image operations such as affine transformation and convolution. The vips processing pipeline has 18 different stages, making therefore the program very parallel and hard to control. In the presented test, an image of 18000×18000 pixels is processed.

The grey box model (3) and (4), having as input only the number of

allotted cores, was identified offline with an LS-based technique on several runs. The identified parameters are

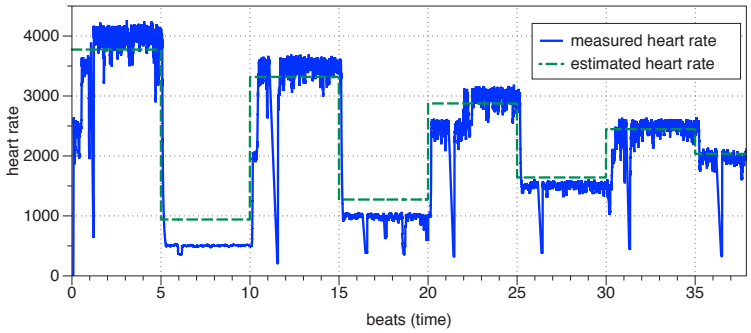
$$\hat{\vartheta}_{vips} = \begin{bmatrix} k_c \\ \alpha_c \\ o_c \end{bmatrix} = \begin{bmatrix} 258.75388 \\ 1.1930687 \\ 681.67218 \end{bmatrix}. \tag{7}$$

An MPC controller was then designed based on an $ARX(1,1)$ (note the orders) model identified on line via RLS. Figure 2 summarises the results. Figure 2a shows that the model reasonably fits the data. The satisfactory adaptive control behaviour in the face of the application behaviour variations, induced by the various phases of its algorithm, is shown in Figures 2b and 2c, that respectively report the desired and actual heart rate, and the control signals. To make the adaptive controller converge to a stable solution, a PI controller – better described in [16] – is used to control the application for the first 100 time units. Also, notice that the process time-variance has been modelled as parameter uncertainty, which causes the peaks in the plots.

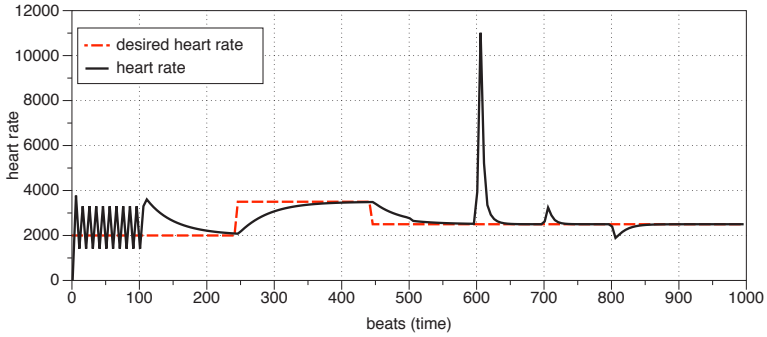
6.1.1 Brief comparison with an alternative identification approach

As noticed, the employed adaptive controller uses a very simple model, suggested by the structure of (3) and the concept of speedup induced by the SMART problem structuring. This would *not* be the case if identification was applied without said problem (and controller) structuring. To witness that, Figure 3 shows how models with different complexity reproduce the measured data without the SMART structuring, i.e., considering as input the number of cores, and as output the heart rate.

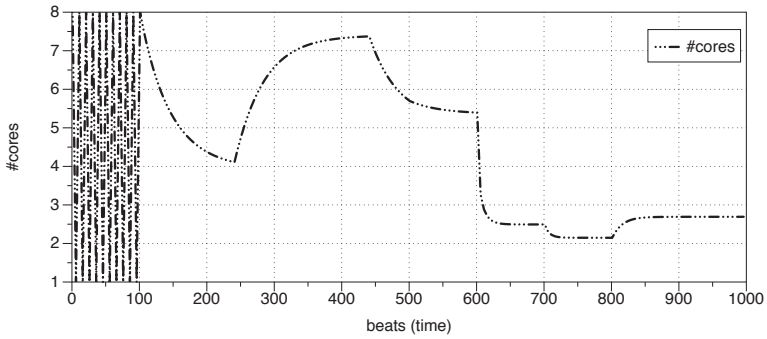
As can be seen, complex models seem to be unconditionally preferable,



(a) Collected data from the system (blue) vs Model simulation (green).



(b) Simulation results of the heart rate (black) set-point following (red).



(c) Simulation results of the control signal (black), which in this case corresponds to the number of cores.

Figure 2: Simulation results for vips control.

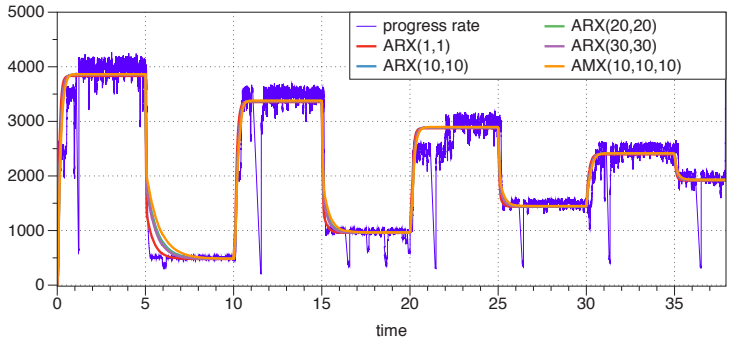


Figure 3: Vips identification results with different model structures.

Table 1: Results obtained in the Matlab Identification Toolbox.

Model	Delay	Best Fits
$ARMAX(10,10,10)$	1	62.24
$ARX(30,30)$	9	61.63
$ARX(20,20)$	9	61.53
$ARX(10,10)$	9	61.36
$ARX(1,1)$	1	58.98

so that typical model order selection tools (the MATLAB `ident` toolbox was used here) invariantly tend to select the maximum allowed model orders, see Table 1, where the fit measure is set to

$$\left[1 - \frac{\|Y - \hat{Y}\|_2}{\|Y - \bar{Y}\|_2}\right] \cdot 100 \tag{8}$$

Such a situation can reasonably be blamed on the absence of the grey box model structuring typical of SMART. In the case at hand, after abstracting the synthetic "speedup" actuator instead of sticking to have as inputs the physical one (number of cores), said approach leads to detect a very small order dynamics cascaded to a substantially static nonlinearity, that an adaptive mechanism based on a very simple ARX can handle satisfactorily.

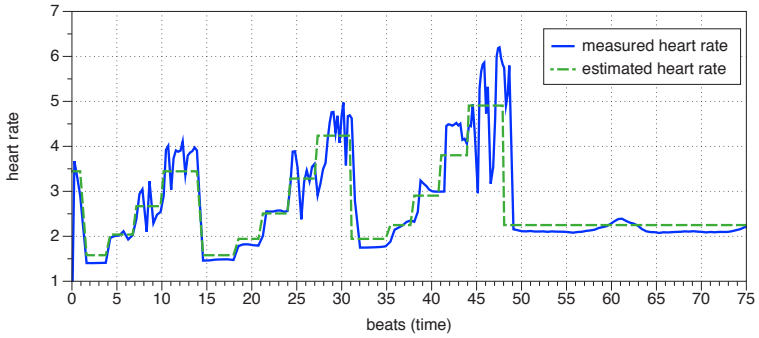
6.2 Example 2 - bodytrack

The bodytrack application tracks the 3D pose of a human body with multiple cameras and no markers, employing an annealed particle filter. It has a parallel thread pool, with three different kernels [4]. The presented tests use 4 cameras, 261 frames, 4000 particles, and 5 annealing layers.

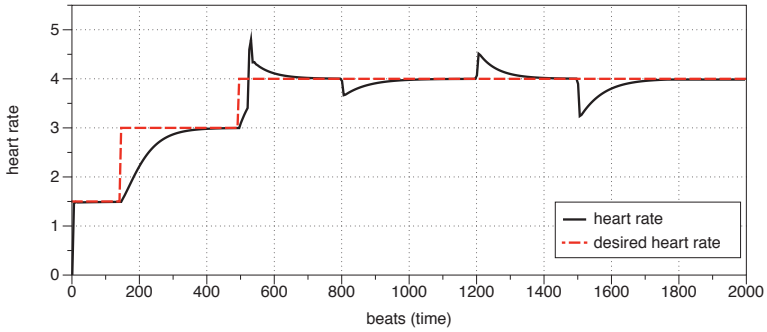
In this case the grey box model 3 and 4, having as inputs the number of allotted cores and the clock frequency, was identified offline as in Section 6.1. The obtained parameters are

$$\hat{\vartheta}_{bodytrack} = \begin{bmatrix} k_c \\ \alpha_c \\ o_c \\ k_f \\ \alpha_f \\ o_f \end{bmatrix} = \begin{bmatrix} 0.1931659 \\ 1.613834 \\ 3.5964752 \\ 2.3736936 \\ 0.1609101 \\ -1.9965658 \end{bmatrix}. \tag{9}$$

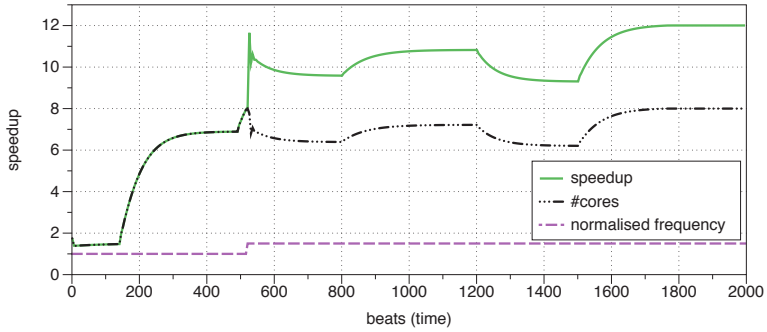
Also in this case, an MPC controller was designed based on an $ARX(1,1)$ (once more, note the orders) model identified on line via RLS. Figure 4, organised in the same way as Figure 2, summarises the results. The satisfactory adaptive control behaviour in the face of the application behaviour variations, induced by the various phases of its algorithm, is shown in the Figure 4b and 4c, that respectively report the desired and actual heart rate, and the control signals. Notice that here too a very simple ARX model is used.



(a) Collected data from the bodytrack software application (blue) and simulation with the identified model (green).



(b) Simulation results of the heart rate (black) set-point following (red).

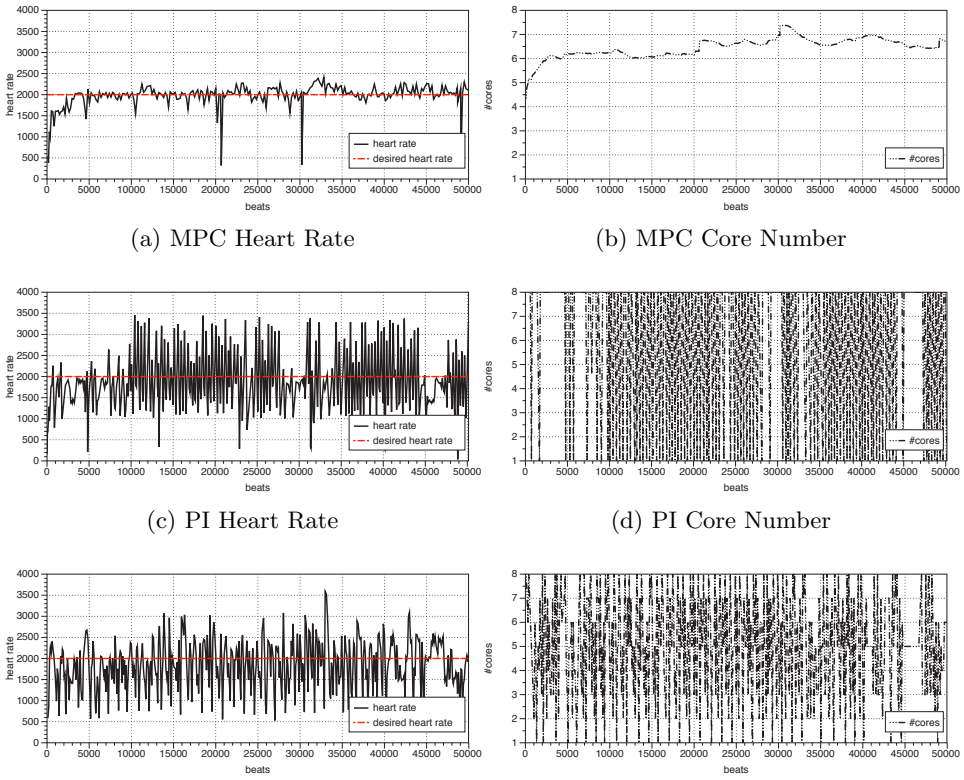


(c) Simulation results of the control signal (green), number of cores (black) and normalised clock frequency (violet).

Figure 4: Bodytrack identification and (simulated) control.

6.3 Experimental verification

In the tests of this section, the two applications are run on a real machine[1] using the number of cores as the only available actuator.



(a) MPC Heart Rate (b) MPC Core Number

(c) PI Heart Rate (d) PI Core Number

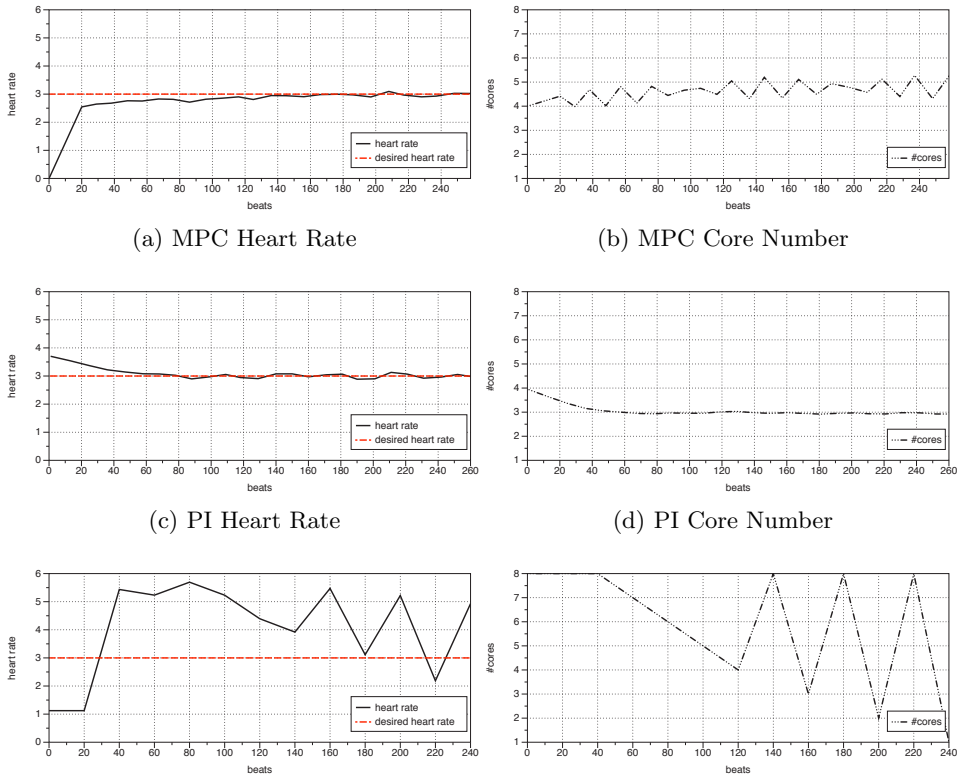
(e) SARSA Heart Rate (f) SARSA Core Number

Figure 5: Vips Experimental Results

To compare the proposed solution with differently conceived alternatives, a PI-based scheme was realised as proposed in [23], and a SARSA reinforcement learning [33] was implemented—details can be found in [24].

As a first example, vips is controlled with an MPC developed within the SMART framework, which produces the results of Figures 5a and 5b. In

[1] All experiments are run on a Dell PowerEdge R410 server with two quad-core Intel Xeon E5530 processors running Linux 2.6.26. The processors support seven power states with clock frequencies from 2.4 GHz to 1.6 GHz.



(a) MPC Heart Rate

(b) MPC Core Number

(c) PI Heart Rate

(d) PI Core Number

(e) SARSA Heart Rate

(f) SARSA Core Number

Figure 6: Bodytrack Experimental Results

the same situation the PI controller behaves as shown in Figures 5c and 5d, and the reinforcement learning algorithm execution is depicted in Figures 5e and 5f.

A second example refers to bodytrack. The SMART MPC controller, developed considering the number of cores as the only available actuator, yields the results of Figures 6a (heart rate) and 6b (number of cores allotted to the application), while the PI controller behaves as shown in Figures 6c and 6d, and the SARSA one as in Figures 6e and 6f.

As can be noticed, some applications are simple to instrument (e.g., bodytrack) and can be managed by more or less any solution with comparable results. In such cases, SMART tends to reduce the controller complexity.

On the other hand, other applications (e.g., vips) are far more complex to instrument reliably, and in such cases only adaptive controllers can achieve good results. In this latter situation, the control synthesis ease provided by the SMART structuring is even stronger a benefit.

7 Conclusion and future work

This manuscript suggests that the benefits of control-theoretical methods in the field of computing systems are dramatically enhanced if the perspective is shifted from controlling already functional systems to designing parts of them as controllers. Such a re-design attitude was shown to provide simpler solutions with respect to classical ones, covering a wider spectrum of cases and allowing for better interpretability and formal assessment.

The presented ideas do not of course apply to any computing system control problem, and their applicability limits were synthetically evidenced. Nonetheless, the variety of the addressable problem makes the proposal interesting, at least in the opinion of the authors.

Moreover, heuristics is confined into a single step of the design procedure and may possible be removed in the future if some suitable set of principles is devised. The quest for such principles will be the subject of future research, the ultimate goal being the realisation of objects such as a totally control-based operating system.

References

[1] T. F. Abdelzaher, Y. Lu, R. Zhang, and D. Henriksson. Practical application of control theory to web services. In *Proceedings of the*

2004 American Control Conference, volume 3, pages 1992–1997, July 2004.

[2] D. H. Albonesi, R. Balasubramonian, S. G. Dropsho, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36:49–58, December 2003.

[3] M. Ben-Yehuda, D. Breitgand, M. Factor, H. Kolodner, V. Kravtsov, and D. Pelleg. Nap: a building block for remediating performance bottlenecks via black box network analysis. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 179–188, New York, NY, USA, 2009. ACM.

[4] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[5] C. Bienia, S. Kumar, J. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008.

[6] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 318–329, Washington, DC, USA, 2008. IEEE Computer Society.

[7] A. Block, B. Brandenburg, J. H. Anderson, and S. Quint. An adaptive framework for multiprocessor real-time system. In *Proceedings of the*

2008 Euromicro Conference on Real-Time Systems, ECRTS '08, pages 23–33, Washington, DC, USA, 2008. IEEE Computer Society.

[8] G. Candea, E. Kiciman, S. Zhang, P. Keyani, and A. Fox. JAGR: an autonomous self-recovering application server. In *Autonomic Computing Workshop*, pages 168–177, June 2003.

[9] C. Cascaval, E. Duesterwald, P. F. Sweeney, and R. W. Wisniewski. Performance and environment monitoring for continuous program optimization. *IBM Journal Research Development*, 50(2/3):239–248, 2006.

[10] S. Choi and D. Yeung. Learning-based SMT processor resource distribution via hill-climbing. In *Proceedings of the 33rd annual international symposium on Computer Architecture*, ISCA '06, pages 239–251, Washington, DC, USA, 2006. IEEE Computer Society.

[11] R. Dantu and J. W. Cangussu. An architecture for network security using feedback control. In *ISI*, pages 636–637, 2005.

[12] R. Dantu, J. W. Cangussu, and S. Patwardhan. Fast worm containment using feedback control. *IEEE Trans. Dependable Secur. Comput.*, 4:119–136, April 2007.

[13] C.-J. Hamann, M. Roitzsch, L. Reuther, J. Wolter, and H. Hartig. Probabilistic admission control to govern real-time systems under overload. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 211–222, Washington, DC, USA, 2007. IEEE Computer Society.

[14] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. Wiley, 2004.

[15] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. Miller, and A. Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *ICAC '10: Proceeding of the 7th international conference on Autonomic computing*, pages 79–88, New York, NY, USA, 2010. ACM.

[16] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Agarwal, and A. Leva. SEEC: a framework for self-aware computing. Technical Report MIT-CSAIL-TR-2010-049, Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, Cambridge, October 2010.

[17] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 39–50, Washington, DC, USA, 2008. IEEE Computer Society.

[18] C. Karamanolis, M. Karlsson, and X. Zhu. Designing controllable computer systems. In *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, pages 9–9, Berkeley, CA, USA, 2005. USENIX Association.

[19] O. Krieger, M. Auslander, B. Rosenburg, R. W. Wisniewski, J. Xenidis, D. Da Silva, M. Ostrowski, J. Appavoo, M. Butrico, M. Mergen, A. Waterland, and V. Uhlig. K42: building a complete operating system. *SIGOPS Oper. Syst. Rev.*, 40:133–145, April 2006.

[20] A. Leva and M. Maggio. Feedback process scheduling with simple discrete-time control structures. *IET Control theory and applications*,

4(11):2331–2342, November 2010.

[21] B. Li and K. Nahrstedt. A control-based middleware framework for quality-of-service adaptations. *IEEE Journal on Selected Areas in Communications*, 17(9):1632 –1650, Sept. 1999.

[22] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23:85–126, July 2002.

[23] M. Maggio, H. Hoffmann, M. Santambrogio, A. Agarwal, and A. Leva. Controlling software applications within the heartbeats framework. In *49th IEEE Conference on Decision and Control*, December 2010.

[24] M. Maggio, H. Hoffmann, M. Santambrogio, A. Agarwal, and A. Leva. Decision making in autonomic computing systems: Comparison of different approaches and techniques. In *ICAC '11: Proceeding of the 7th international conference on Autonomic computing*, Karlsrhue, Germany, 2011. ACM.

[25] M. Maggio and A. Leva. A new perspective proposal for preemptive feedback scheduling. *International Journal of Innovative Computing, Information and Control*, 6(6), October 2010.

[26] M. Maggio and A. Leva. Toward a deeper use of feedback control in the design of critical computing system components. In *49th IEEE Conference on Decision and Control*, pages 5985–5990, 2010.

[27] M. Maggio, A. Papadopoulos, and A. Leva. On the use of feedback control in the design of computing system components. *Asian Journal of Control*. (in press).

[28] M. Maggio, F. Terraneo, and A. Leva. Implementation and evaluation of a control-theoretical scheduler. *ICIC Express Letters*, 4(6b):2343–2347, December 2010.

[29] S. Oberthür, C. Böke, and B. Griese. Dynamic online reconfiguration for customizable and self-optimizing operating systems. In *Proceedings of the 5th ACM international conference on Embedded software*, EMSOFT '05, pages 335–338, New York, NY, USA, 2005. ACM.

[30] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 289–302, New York, NY, USA, 2007. ACM.

[31] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher. Queueing model based network server performance control. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, RTSS '02, pages 81–, Washington, DC, USA, 2002. IEEE Computer Society.

[32] Q. Sun, G. Dai, and W. Pan. LPV model and its application in web server performance control. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 03*, CSSE '08, pages 486–489, Washington, DC, USA, 2008. IEEE Computer Society.

[33] R. S. Sutton and A. G. Barto. *Reinforcement Learning: an introduction*. Adaptive Computation and Machine Learning series. MIT Press (Bradford Book), Cambridge, MA, USA, 1998.

[34] M. Tanelli, D. Ardagna, and M. Lovera. LPV model identification for power management of web service systems. In *Control Applications, 2008. CCA 2008. IEEE International Conference on*, pages 1171 –1176, September 2008.

[35] T. Voigt and P. Gunningberg. Adaptive resource-based web server admission control. In *Proceedings of the 7th International Symposium on Computers and Communications*, pages 219–224, 2002.

[36] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, pages 301–306, Washington, DC, USA, 2002. IEEE Computer Society.